ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

Лабораторная №3

По дисциплине «Архитектура вычислительных систем»

«Оценка производительности процессора»

Выполнил: студент гр. ИП-013

Копытина Т.А.

Проверил: ассистент кафедры ВС

Насонова А.О.

Новосибирск 2022г.

Оглавление

Цель лабораторной работы

Разработать программу (benchmark) для оценки производительности подсистемы памяти.

Описание задачи:

1. Написать программу(функцию)на языке C/C++/C#для оценки производительности подсистемы памяти.

На вход программы подать следующие аргументы.

- Подсистема памяти. Предусмотреть возможность указать подсистему для проверки производительности: RAM (оперативная память), HDD/SSD и flash.
- Размер блока данных в байтах, Кб или Мб. Если размерность не указана, то в байтах, если указана, то соответственно в Кбайтах или Мбайтах.
- Число испытаний, т.е. число раз повторений измерений.

2. Написать программу(функцию) на языке C/C++/C# или скрипт, (benchmark) реализующий серию испытаний программы(функции) из п.1. Оценить пропускную способность оперативной памяти при работе с блоками данных равными объёму кэш-линии, кэш-памяти L1, L2 и L3 уровня и превышающего его. Для HDD|SSD и flash провести серию из 20 испытаний с блоками данных начиная с 4 Мб с шагом 4Мб. Результаты всех испытаний сохранить в один CSV файл со структурой, описанной в п.1.
3. На основе CSV файла построить сводные таблицы и диаграммы, отражающие:

- Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для разного типа памяти;
- Зависимость погрешности измерения пропускной способности от размера блока данных для разного типа памяти;

- Зависимость погрешности измерений от числа испытаний LaunchNum;

Ход работы

Вначале работы напишем `mem_t.cpp, mem_t.hpp, main, mem_1.cpp` файлы, в которых опишем функции для работы.

В `mem_1.cpp` опишем три функции:

1. `int code_mem(string mem_t)` в данной функции опишем возвращения значения переменной `mem_t` . Данная переменная может вернуть 0 если равна RAM, 1 если равна HDD, 2 если равна flash, если же функция не равна ни одному из этих значений, то она вернет ошибку.
2. Функция `int calc_size(int size_bl, string unit)` будет вычислять размер данным в байтах, Кб или Мб.
3. Последнюю функцию опишем основную `int main(int argc, char *argv[])`.

В файле `mem_t` опишем 5 функций:

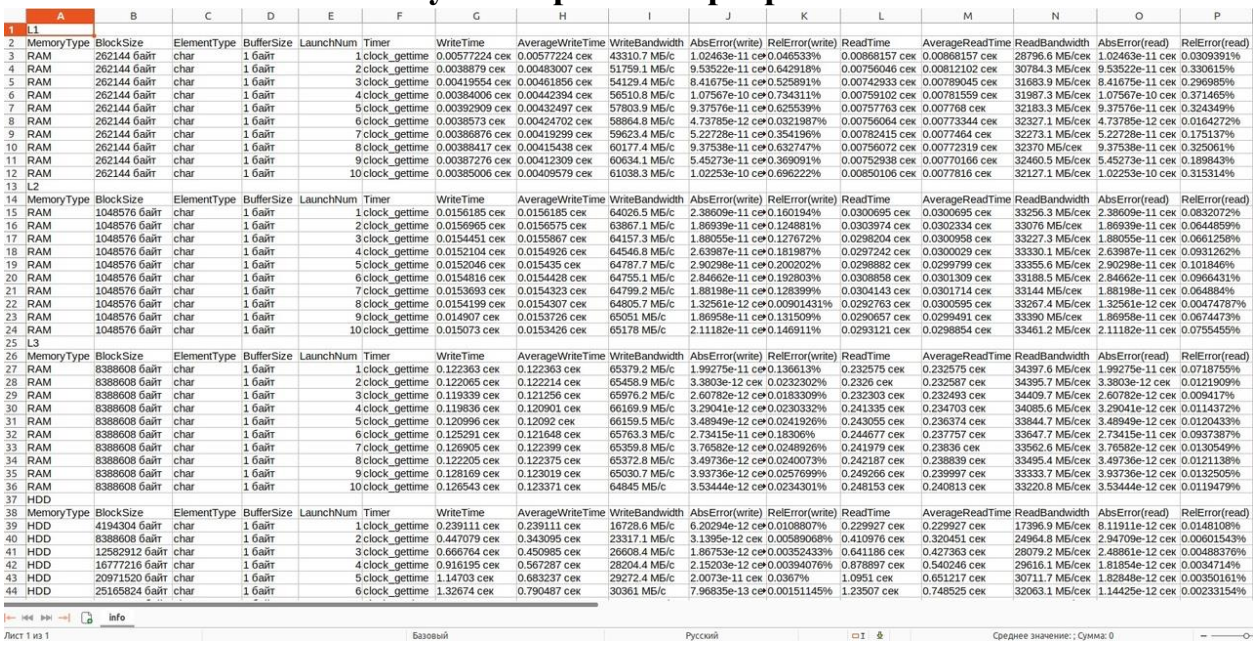
1. Первой опишем функцию тестирования RAM памяти `void ram_test(int size_bl)`
2. Второй опишем функцию тестирования HDD памяти `void hdd_ssd_test(int size_bl)`
3. Третья функция тестирования flash памяти `void flash_test(int size_bl, string path)`
4. Четвертая функция записи тестов в csv файл `void start_test(int N, int type_mem, int launch, string message, bool fl_ser)`
5. Пятая функция – это функция записи результатов пропускной способности памяти `void write_result(int type_mem, int launch, int bl_size, ofstream &out)`

В файле `mem_t.hpp` опишем все функции, которые находятся в файле `mem_t.cpp`.

В файле `main.cpp` опишем 6 функций:

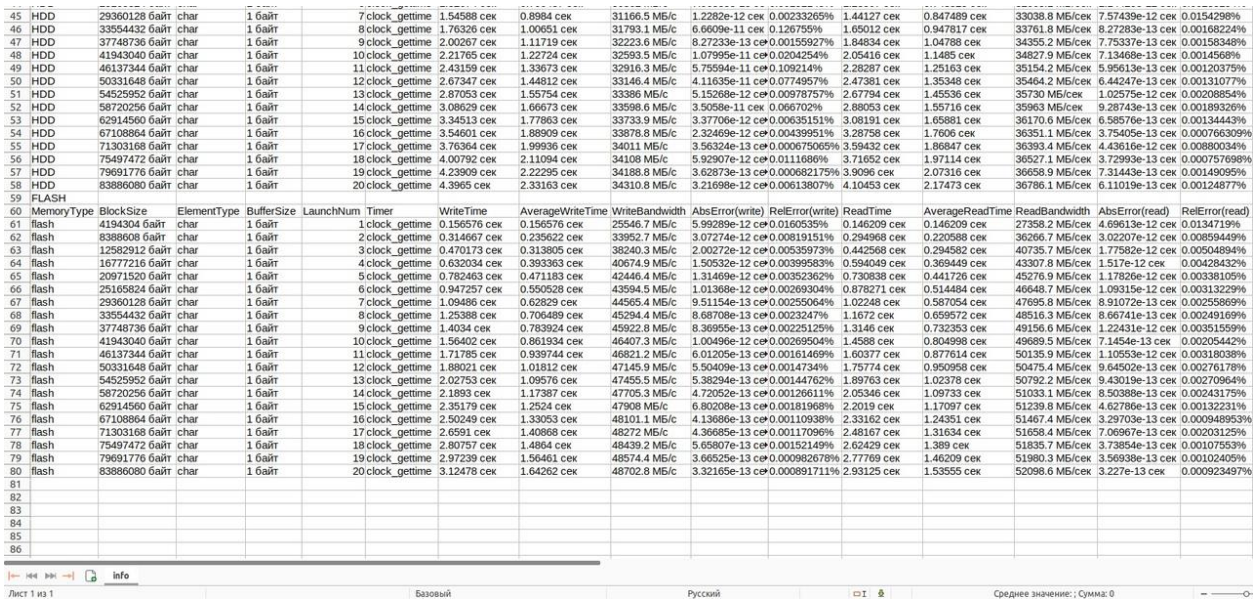
1. Первой опишем функцию `void DGEMM(double **a, double **b, double **c, long long N)` . Вызов подпрограммы DGEMM умножает матрицы с числами вещественного типа.
2. Второй опишем функцию `void DGEMM_OPT1(double **a, double **b, double **c, long long N)`. Она выполняет те же действия что и предыдущая функция.
3. Третья функция `void DGEMM_OPT2(double **a, double **b, double **c, long long N)` делает все то же.
4. Четвертая функция выполняет случайное заполнения двух матриц `void rand_mass(double **a, double **b, long long n)`
5. Пятая функция `void series_func(string multiply, int left, int right)` выполняет запись результатов перемножения в файлы result_opt1.txt и result_opt2.txt.
6. И последней напишем функцию `void single_func(string multiply, int n)`.

Результат работы программы



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	L1															
2	MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
3	RAM	262144 байт	char	1 байт	1	clock_gettime	0.00577224 сек	0.00577224 сек	43310.7 МБ/с	1.02463e-11 се	0.046533%	0.00868157 сек	0.00868157 сек	28796.6 МБ/сек	1.02463e-11 сек	0.0309391%
4	RAM	262144 байт	char	1 байт	2	clock_gettime	0.0038879 сек	0.00483007 сек	51759.1 МБ/с	9.53522e-11 се	0.642918%	0.00756046 сек	0.00812102 сек	30784.3 МБ/сек	9.53522e-11 сек	0.330615%
5	RAM	262144 байт	char	1 байт	3	clock_gettime	0.00419554 сек	0.00461856 сек	54129.4 МБ/с	8.41675e-11 се	0.525891%	0.00742933 сек	0.00789045 сек	31683.9 МБ/сек	8.41675e-11 сек	0.296985%
6	RAM	262144 байт	char	1 байт	4	clock_gettime	0.00384006 сек	0.00442394 сек	56510.8 МБ/с	1.07567e-10 се	0.734311%	0.00759102 сек	0.00781559 сек	31987.3 МБ/сек	1.07567e-10 сек	0.371465%
7	RAM	262144 байт	char	1 байт	5	clock_gettime	0.00392909 сек	0.00432497 сек	57803.9 МБ/с	9.37576e-11 се	0.625539%	0.00757763 сек	0.007768 сек	32183.3 МБ/сек	9.37576e-11 сек	0.324349%
8	RAM	262144 байт	char	1 байт	6	clock_gettime	0.0038573 сек	0.00424702 сек	58864.8 МБ/с	4.73785e-12 се	0.0321987%	0.00756064 сек	0.00773344 сек	32327.1 МБ/сек	4.73785e-12 сек	0.0164272%
9	RAM	262144 байт	char	1 байт	7	clock_gettime	0.00386876 сек	0.00419299 сек	59623.4 МБ/с	5.22728e-11 се	0.354196%	0.00782415 сек	0.0077464 сек	32273.1 МБ/сек	5.22728e-11 сек	0.175137%
10	RAM	262144 байт	char	1 байт	8	clock_gettime	0.00388417 сек	0.00415438 сек	60177.4 МБ/с	9.37538e-11 се	0.632747%	0.00756072 сек	0.00772319 сек	32370 МБ/сек	9.37538e-11 сек	0.325061%
11	RAM	262144 байт	char	1 байт	9	clock_gettime	0.00387276 сек	0.00412309 сек	60634.1 МБ/с	5.45273e-11 се	0.369091%	0.00752938 сек	0.00770166 сек	32460.5 МБ/сек	5.45273e-11 сек	0.189843%
12	RAM	262144 байт	char	1 байт	10	clock_gettime	0.00385006 сек	0.00409579 сек	61038.3 МБ/с	1.02253e-10 се	0.696222%	0.00850106 сек	0.0077816 сек	32127.1 МБ/сек	1.02253e-10 сек	0.315314%
13	L2															
14	MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
15	RAM	1048576 байт	char	1 байт	1	clock_gettime	0.0156185 сек	0.0156185 сек	64026.5 МБ/с	2.38609e-11 се	0.160194%	0.0300695 сек	0.0300695 сек	33256.3 МБ/сек	2.38609e-11 сек	0.0832072%
16	RAM	1048576 байт	char	1 байт	2	clock_gettime	0.0156965 сек	0.0156575 сек	63867.1 МБ/с	1.86939e-11 се	0.124881%	0.0303974 сек	0.0302334 сек	33076 МБ/сек	1.86939e-11 сек	0.0644859%
17	RAM	1048576 байт	char	1 байт	3	clock_gettime	0.0154451 сек	0.0155867 сек	64157.3 МБ/с	1.88055e-11 се	0.127672%	0.0298204 сек	0.0300958 сек	33227.3 МБ/сек	1.88055e-11 сек	0.0661258%
18	RAM	1048576 байт	char	1 байт	4	clock_gettime	0.0152104 сек	0.0154926 сек	64546.8 МБ/с	2.63987e-11 се	0.181987%	0.0297242 сек	0.0300029 сек	33330.1 МБ/сек	2.63987e-11 сек	0.0931262%
19	RAM	1048576 байт	char	1 байт	5	clock_gettime	0.0152046 сек	0.015435 сек	64787.7 МБ/с	2.90298e-11 се	0.200202%	0.0298882 сек	0.0299799 сек	33355.6 МБ/сек	2.90298e-11 сек	0.101846%
20	RAM	1048576 байт	char	1 байт	6	clock_gettime	0.0154816 сек	0.0154428 сек	64755.1 МБ/с	2.84662e-11 се	0.192803%	0.0308858 сек	0.0301309 сек	33188.5 МБ/сек	2.84662e-11 сек	0.0966431%
21	RAM	1048576 байт	char	1 байт	7	clock_gettime	0.0153693 сек	0.0154323 сек	64799.2 МБ/с	1.88198e-11 се	0.128399%	0.0304143 сек	0.0301714 сек	33144 МБ/сек	1.88198e-11 сек	0.064884%
22	RAM	1048576 байт	char	1 байт	8	clock_gettime	0.0154199 сек	0.0154307 сек	64805.7 МБ/с	1.32561e-12 се	0.00901431%	0.0292763 сек	0.0300595 сек	33267.4 МБ/сек	1.32561e-12 сек	0.0047487%
23	RAM	1048576 байт	char	1 байт	9	clock_gettime	0.014907 сек	0.0153726 сек	65051 МБ/с	1.86958e-11 се	0.131509%	0.0290657 сек	0.0299491 сек	33390 МБ/сек	1.86958e-11 сек	0.0674473%
24	RAM	1048576 байт	char	1 байт	10	clock_gettime	0.015073 сек	0.0153426 сек	65178 МБ/с	2.11182e-11 се	0.146911%	0.0293121 сек	0.0298854 сек	33461.2 МБ/сек	2.11182e-11 сек	0.0755455%
25	L3															
26	MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
27	RAM	8388608 байт	char	1 байт	1	clock_gettime	0.122363 сек	0.122363 сек	65379.2 МБ/с	1.99275e-11 се	0.13661%	0.232575 сек	0.232575 сек	34397.6 МБ/сек	1.99275e-11 сек	0.0718755%
28	RAM	8388608 байт	char	1 байт	2	clock_gettime	0.122065 сек	0.122214 сек	65458.9 МБ/с	3.3803e-12 сек	0.0232302%	0.2326 сек	0.232587 сек	34395.7 МБ/сек	3.3803e-12 сек	0.0121909%
29	RAM	8388608 байт	char	1 байт	3	clock_gettime	0.119339 сек	0.121256 сек	65976.2 МБ/с	2.60782e-12 се	0.0183309%	0.232303 сек	0.232493 сек	34409.7 МБ/сек	2.60782e-12 сек	0.009417%
30	RAM	8388608 байт	char	1 байт	4	clock_gettime	0.119836 сек	0.120901 сек	66169.9 МБ/с	3.29041e-12 се	0.0230332%	0.241335 сек	0.234703 сек	34085.6 МБ/сек	3.29041e-12 сек	0.0114372%
31	RAM	8388608 байт	char	1 байт	5	clock_gettime	0.120996 сек	0.12092 сек	66159.5 МБ/с	3.48949e-12 се	0.0241926%	0.243055 сек	0.236374 сек	33844.7 МБ/сек	3.48949e-12 сек	0.0120433%
32	RAM	8388608 байт	char	1 байт	6	clock_gettime	0.125291 сек	0.121648 сек	65763.3 МБ/с	2.73415e-11 се	0.18306%	0.244677 сек	0.237757 сек	33647.7 МБ/сек	2.73415e-11 сек	0.0937387%
33	RAM	8388608 байт	char	1 байт	7	clock_gettime	0.126905 сек	0.122399 сек	65359.8 МБ/с	3.76582e-12 се	0.0248926%	0.241979 сек	0.23836 сек	33562.6 МБ/сек	3.76582e-12 сек	0.0130549%
34	RAM	8388608 байт	char	1 байт	8	clock_gettime	0.122205 сек	0.122375 сек	65372.8 МБ/с	3.49736e-12 се	0.0240073%	0.242187 сек	0.238839 сек	33495.4 МБ/сек	3.49736e-12 сек	0.0121138%
35	RAM	8388608 байт	char	1 байт	9	clock_gettime	0.128169 сек	0.123019 сек	65030.7 МБ/с	3.93736e-12 се	0.0257699%	0.249266 сек	0.239997 сек	33333.7 МБ/сек	3.93736e-12 сек	0.0132505%
36	RAM	8388608 байт	char	1 байт	10	clock_gettime	0.126543 сек	0.123371 сек	64845 МБ/с	3.53444e-12 се	0.0234301%	0.248153 сек	0.240813 сек	33220.8 МБ/сек	3.53444e-12 сек	0.0119479%
37	HDD															
38	MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
39	HDD	4194304 байт	char	1 байт	1	clock_gettime	0.239111 сек	0.239111 сек	16728.6 МБ/с	6.20294e-12 се	0.0108807%	0.229927 сек	0.229927 сек	17396.9 МБ/сек	8.11911e-12 сек	0.0148108%
40	HDD	8388608 байт	char	1 байт	2	clock_gettime	0.447079 сек	0.343095 сек	23317.1 МБ/с	3.1395e-12 сек	0.00589068%	0.410976 сек	0.320451 сек	24964.8 МБ/сек	2.94709e-12 сек	0.00601543%
41	HDD	12582912 байт	char	1 байт	3	clock_gettime	0.666764 сек	0.450985 сек	26608.4 МБ/с	1.86753e-12 се	0.00352433%	0.641186 сек	0.427363 сек	28079.2 МБ/сек	2.48861e-12 сек	0.00488376%
42	HDD	16777216 байт	char	1 байт	4	clock_gettime	0.916195 сек	0.567287 сек	28204.4 МБ/с	2.15203e-12 се	0.00394076%	0.878897 сек	0.540246 сек	29616.1 МБ/сек	1.81854e-12 сек	0.003471%
43	HDD	20971520 байт	char	1 байт	5	clock_gettime	1.14703 сек	0.683237 сек	29272.4 МБ/с	2.0073e-11 сек	0.0367%	1.0951 сек	0.651217 сек	30711.7 МБ/сек	1.82848e-12 сек	0.00350161%
44	HDD	25165824 байт	char	1 байт	6	clock_gettime	1.32674 сек	0.790487 сек	30361 МБ/с	7.96835e-13 се	0.00151145%	1.23507 сек	0.748525 сек	32063.1 МБ/сек	1.14425e-12 сек	0.00233154%

Рисунок 1. Вывод результатов работы в CSVфайл.



45	HDD	29360128 байт	char	1 байт	7	clock_gettime	1.54588 сек	0.8984 сек	31166.5 МБ/с	1.2282e-12 сек	0.00233265%	1.44127 сек	0.847489 сек	33038.8 МБ/сек	7.57439e-12 сек	0.0154298%
46	HDD	33554432 байт	char	1 байт	8	clock_gettime	1.76326 сек	1.00651 сек	31793.1 МБ/с	6.6609e-11 сек	0.126755%	1.65012 сек	0.947817 сек	33761.8 МБ/сек	8.27283e-13 сек	0.00168224%
47	HDD	37748736 байт	char	1 байт	9	clock_gettime	2.00267 сек	1.11719 сек	32223.6 МБ/с	8.27233e-13 се	0.00155927%	1.84834 сек	1.04788 сек	34355.2 МБ/сек	7.75337e-13 сек	0.00158348%
48	HDD	41943040 байт	char	1 байт	10	clock_gettime	2.21765 сек	1.22724 сек	32593.5 МБ/с	1.07995e-11 се	0.0204254%	2.05416 сек	1.1485 сек	34827.9 МБ/сек	7.13468e-13 сек	0.0014568%
49	HDD	46137344 байт	char	1 байт	11	clock_gettime	2.43159 сек	1.33673 сек	32916.3 МБ/с	5.75594e-11 се	0.109214%	2.28287 сек	1.25163 сек	35154.2 МБ/сек	5.95613e-13 сек	0.00120375%
50	HDD	50331648 байт	char	1 байт	12	clock_gettime	2.67347 сек	1.44812 сек	33146.4 МБ/с	4.11635e-11 се	0.0774957%	2.47381 сек	1.35348 сек	35464.2 МБ/сек	6.44247e-13 сек	0.00131077%
51	HDD	54525952 байт	char	1 байт	13	clock_gettime	2.87053 сек	1.55754 сек	33386 МБ/с	5.15268e-12 се	0.00978757%	2.67794 сек	1.45536 сек	35730 МБ/сек	1.02575e-12 сек	0.00208854%
52	HDD	58720256 байт	char	1 байт	14	clock_gettime	3.08629 сек	1.66673 сек	33598.6 МБ/с	3.5058e-11 сек	0.066702%	2.88053 сек	1.55716 сек	35963 МБ/сек	9.28743e-13 сек	0.00189326%
53	HDD	62914560 байт	char	1 байт	15	clock_gettime	3.34513 сек	1.77863 сек	33733.9 МБ/с	3.37706e-12 се	0.00635151%	3.08191 сек	1.65881 сек	36170.6 МБ/сек	6.58576e-13 сек	0.00134443%
54	HDD	67108864 байт	char	1 байт	16	clock_gettime	3.54601 сек	1.88909 сек	33878.8 МБ/с	2.32469e-12 се	0.00439951%	3.28758 сек	1.7606 сек	36351.1 МБ/сек	3.75405e-13 сек	0.0007663094%
55	HDD	71303168 байт	char	1 байт	17	clock_gettime	3.76364 сек	1.99936 сек	34011 МБ/с	3.56324e-13 се	0.0006750656%	3.59432 сек	1.86847 сек	36393.4 МБ/сек	4.43616e-12 сек	0.00880034%
56	HDD	75497472 байт	char	1 байт	18	clock_gettime	4.00792 сек	2.11094 сек	34108 МБ/с	5.92907e-12 се	0.0111686%	3.71652 сек	1.97114 сек	36527.1 МБ/сек	3.72993e-13 сек	0.0007576986%
57	HDD	79691776 байт	char	1 байт	19	clock_gettime	4.23909 сек	2.22295 сек	34188.8 МБ/с	3.62873e-13 се	0.0006821759%	3.9096 сек	2.07316 сек	36658.9 МБ/сек	7.31443e-13 сек	0.00149095%
58	HDD	83886080 байт	char	1 байт	20	clock_gettime	4.3965 сек	2.33163 сек	34310.8 МБ/с	3.21698e-12 се	0.00613807%	4.10453 сек	2.17473 сек	36786.1 МБ/сек	6.11019e-13 сек	0.00124877%
59	FLASH															
60	MemoryType	BlockSize	ElementType	BufferSize	LaunchNum	Timer	WriteTime	AverageWriteTime	WriteBandwidth	AbsError(write)	RelError(write)	ReadTime	AverageReadTime	ReadBandwidth	AbsError(read)	RelError(read)
61	flash	4194304 байт	char	1 байт	1	clock_gettime	0.156576 сек	0.156576 сек	25546.7 МБ/с	5.99289e-12 се	0.0160535%	0.146209 сек	0.146209 сек	27358.2 МБ/сек	4.69613e-12 сек	0.0134719%
62	flash	8388608 байт	char	1 байт	2	clock_gettime	0.314667 сек	0.235622 сек	33952.7 МБ/с	3.07274e-12 се	0.00819151%	0.294968 сек	0.220588 сек	36266.7 МБ/сек	3.02207e-12 сек	0.00859449%
63	flash	12582912 байт	char	1 байт	3	clock_gettime	0.470173 сек	0.313805 сек	38240.3 МБ/с	2.00272e-12 се	0.00535973%	0.442568 сек	0.294582 сек	40735.7 МБ/сек	1.77582e-12 сек	0.00504894%
64	flash	16777216 байт	char	1 байт	4	clock_gettime	0.632034 сек	0.393363 сек	40674.9 МБ/с	1.50532e-12 се	0.00399583%	0.594049 сек	0.369449 сек	43307.8 МБ/сек	1.517e-12 сек	0.00428432%
65	flash	20971520 байт	char	1 байт	5	clock_gettime	0.782463 сек	0.471183 сек	42446.4 МБ/с	1.31469e-12 се	0.00352362%	0.730838 сек	0.441726 сек	45276.9 МБ/сек	1.17826e-12 сек	0.00338105%
66	flash	25165824 байт	char	1 байт	6	clock_gettime	0.947257 сек	0.550528 сек	43594.5 МБ/с	1.01368e-12 се	0.00269304%	0.878271 сек	0.514484 сек	46648.7 МБ/сек	1.09315e-12 сек	0.00313229%
67	flash	29360128 байт	char	1 байт	7	clock_gettime	1.09486 сек	0.62829 сек	44565.4 МБ/с	9.51154e-13 се	0.00255064%	1.02248 сек	0.587054 сек	47695.8 МБ/сек	8.91072e-13 сек	0.00255869%
68	flash	33554432 байт	char	1 байт	8	clock_gettime	1.25388 сек	0.706489 сек	45294.4 МБ/с	8.68708e-13 се	0.0023247%	1.1672 сек	0.659572 сек	48516.3 МБ/сек	8.66741e-13 сек	0.00249169%
69	flash	37748736 байт	char	1 байт	9	clock_gettime	1.4034 сек	0.783924 сек	45922.8 МБ/с	8.36955e-13 се	0.00225125%	1.3146 сек	0.732353 сек	49156.6 МБ/сек	1.22431e-12 сек	0.00351559%
70	flash	41943040 байт	char	1 байт	10	clock_gettime	1.56402 сек	0.861934 сек	46407.3 МБ/с	1.00496e-12 се	0.00269504%	1.4588 сек	0.804998 сек	49689.5 МБ/сек	7.1454e-13 сек	0.00205442%
71	flash	46137344 байт	char	1 байт	11	clock_gettime	1.71785 сек	0.939744 сек	46821.2 МБ/с	6.01205e-13 се	0.00161469%	1.60377 сек	0.877614 сек	50135.9 МБ/сек	1.10553e-12 сек	0.00318038%
72	flash	50331648 байт	char	1 байт	12	clock_gettime	1.88021 сек	1.01812 сек	47145.9 МБ/с	5.50409e-13 се	0.0014734%	1.75774 сек	0.950958 сек	50475.4 МБ/сек	9.64502e-13 сек	0.00276178%
73	flash	54525952 байт	char	1 байт	13	clock_gettime	2.02753 сек	1.09576 сек	47455.5 МБ/с	5.38294e-13 се	0.00144762%	1.89763 сек	1.02378 сек	50792.2 МБ/сек	9.43019e-13 сек	0.00270964%
74	flash	58720256 байт	char	1 байт	14	clock_gettime	2.1893 сек	1.17387 сек	47705.3 МБ/с	4.72052e-13 се	0.00126611%	2.05346 сек	1.09733 сек	51033.1 МБ/сек	8.50388e-13 сек	0.00243175%
75	flash	62914560 байт	char	1 байт	15	clock_gettime	2.35179 сек	1.2524 сек	47908 МБ/с	6.80208e-13 се	0.00181968%	2.2019 сек	1.17097 сек	51239.8 МБ/сек	4.62786e-13 сек	0.00132231%
76	flash	67108864 байт	char	1 байт	16	clock_gettime	2.50249 сек	1.33053 сек	48101.1 МБ/с	4.13686e-13 се	0.00110938%	2.33162 сек	1.24351 сек	51467.4 МБ/сек	3.29703e-13 сек	0.0009489534%
77	flash	71303168 байт	char	1 байт	17	clock_gettime	2.6591 сек	1.40868 сек	48272 МБ/с	4.36685e-13 се	0.00117096%	2.48167 сек	1.31634 сек	51658.4 МБ/сек	7.06967e-13 сек	0.00203125%
78	flash	75497472 байт	char	1 байт	18	clock_gettime	2.80757 сек	1.4864 сек	48439.2 МБ/с	5.65807e-13 се	0.00152149%	2.62429 сек	1.389 сек	51835.7 МБ/сек	3.73854e-13 сек	0.00107553%
79	flash	79691776 байт	char	1 байт	19	clock_gettime	2.97239 сек	1.56461 сек	48574.4 МБ/с	3.66525e-13 се	0.000982678%	2.77769 сек	1.46209 сек	51980.3 МБ/сек	3.56938e-13 сек	0.00102405%
80	flash	83886080 байт	char	1 байт	20	clock_gettime	3.12478 сек	1.64262 сек	48702.8 МБ/с	3.32165e-13 се	0.000891711%	2.93125 сек	1.53555 сек	52098.6 МБ/сек	3.227e-13 сек	0.0009234974%
81																
82																
83																
84																
85																
86																

Рисунок 2. Вывод результатов работы в CSV файл.

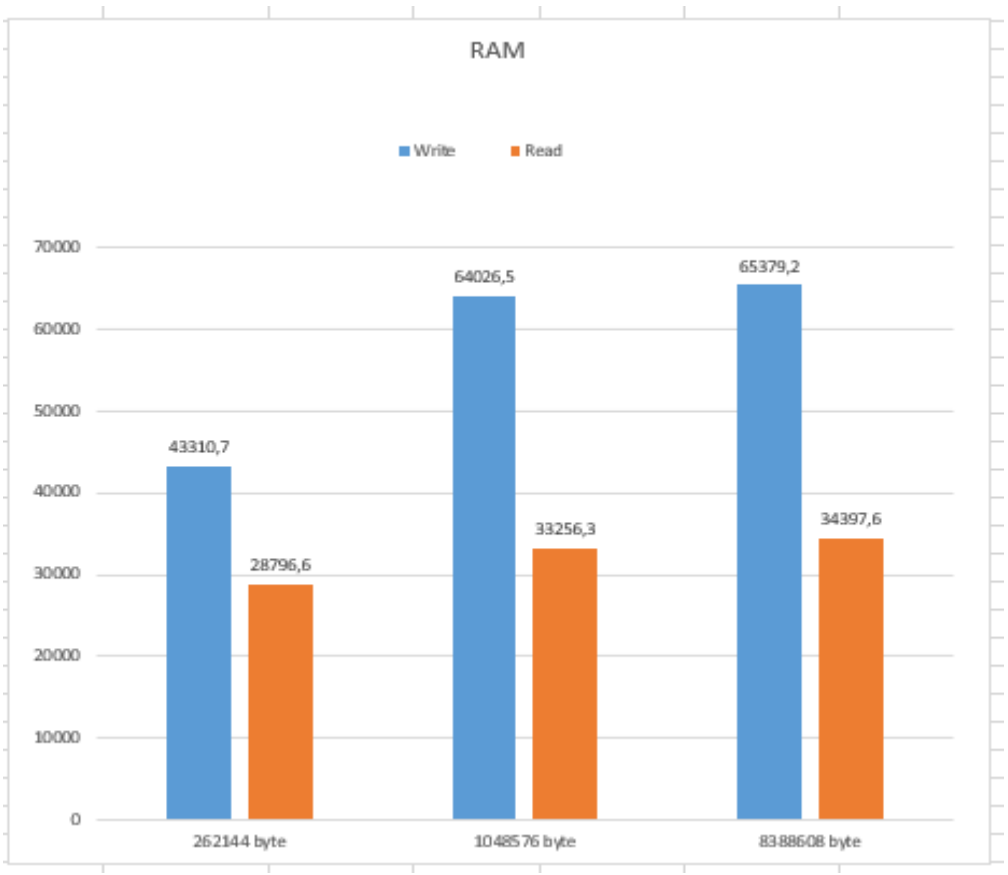


Рисунок 3. Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для RAM памяти;

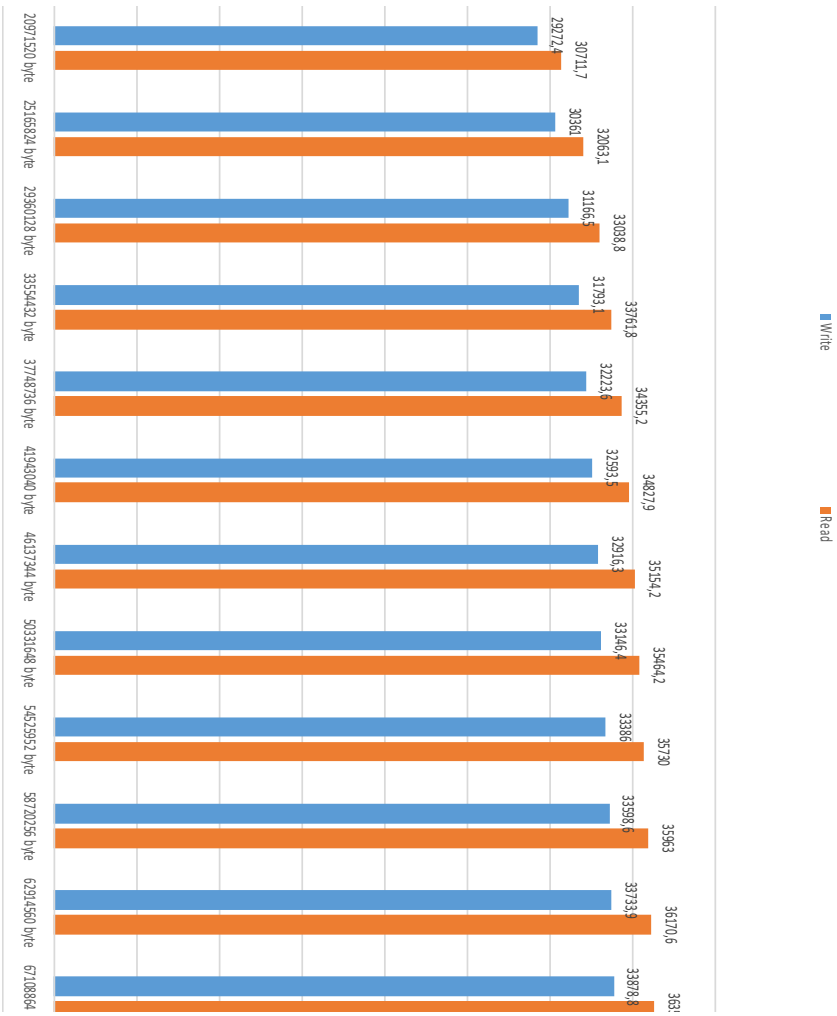


Рисунок 4. Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для HDD памяти

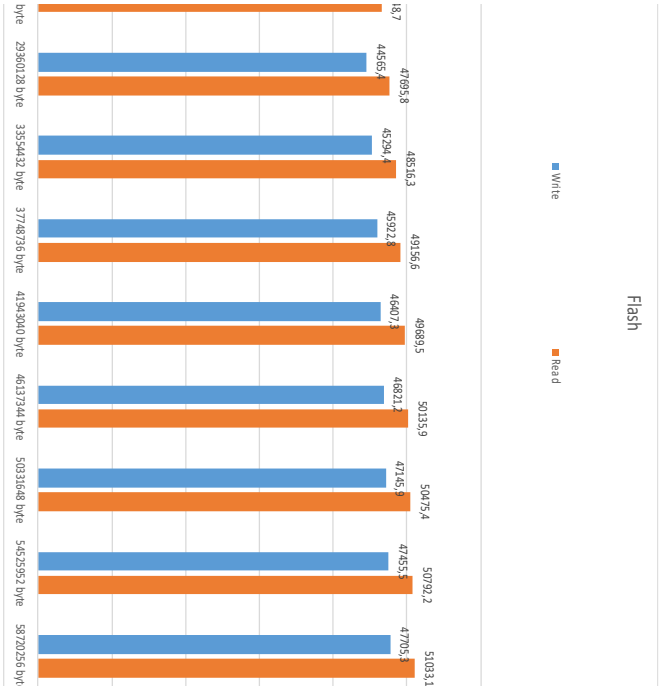


Рисунок 5. Зависимость пропускной способности записи и чтения от размера блока данных (BlockSize) для flash памяти

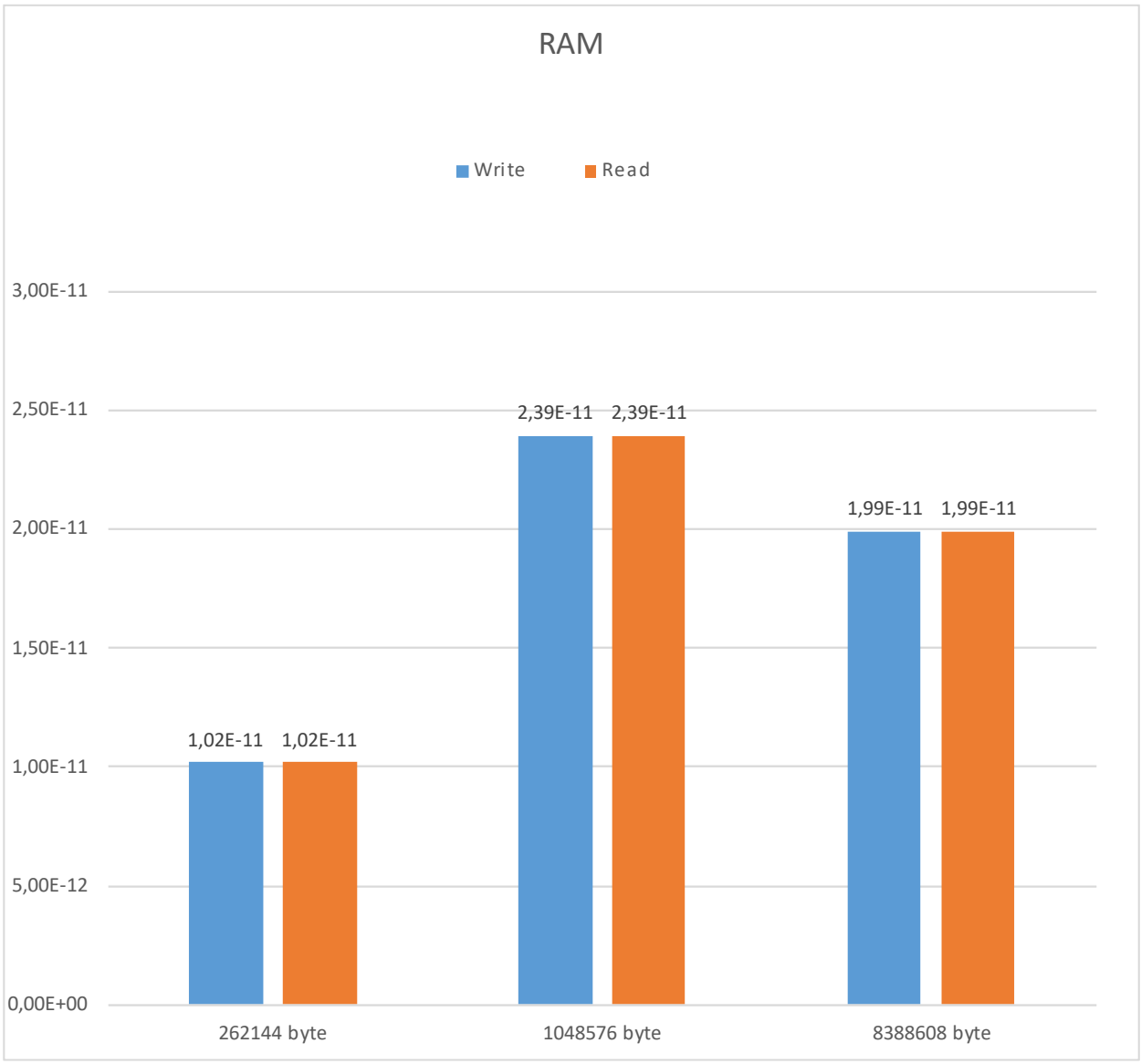


Рисунок 6. Зависимость погрешности измерения пропускной способности от размера блока данных для RAM памяти.

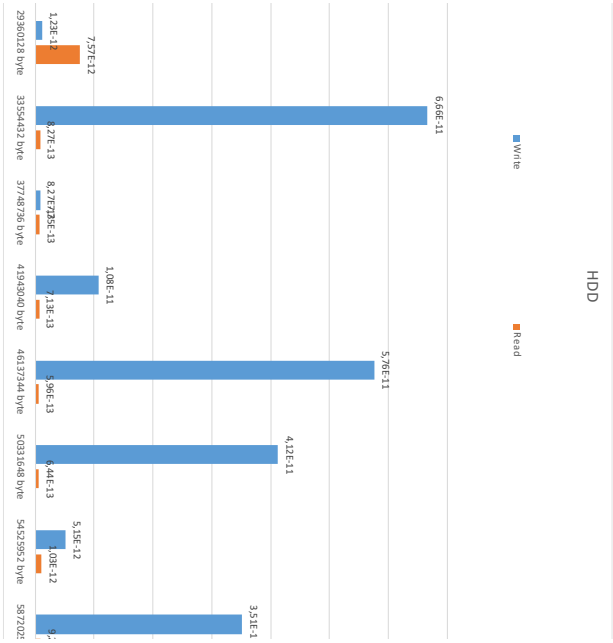


Рисунок 7. Зависимость погрешности измерения пропускной способности от размера блока данных для HDD памяти

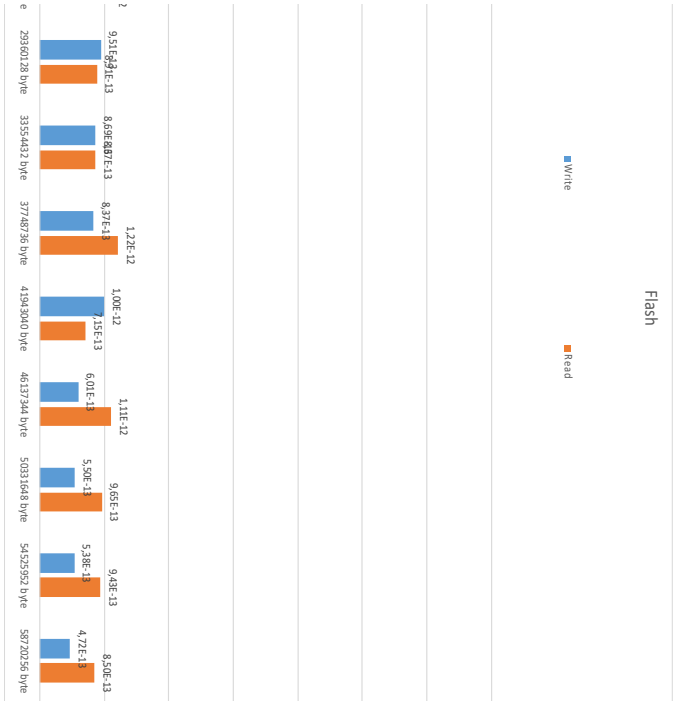


Рисунок 8. Зависимость погрешности измерения пропускной способности от размера блока данных для flash памяти

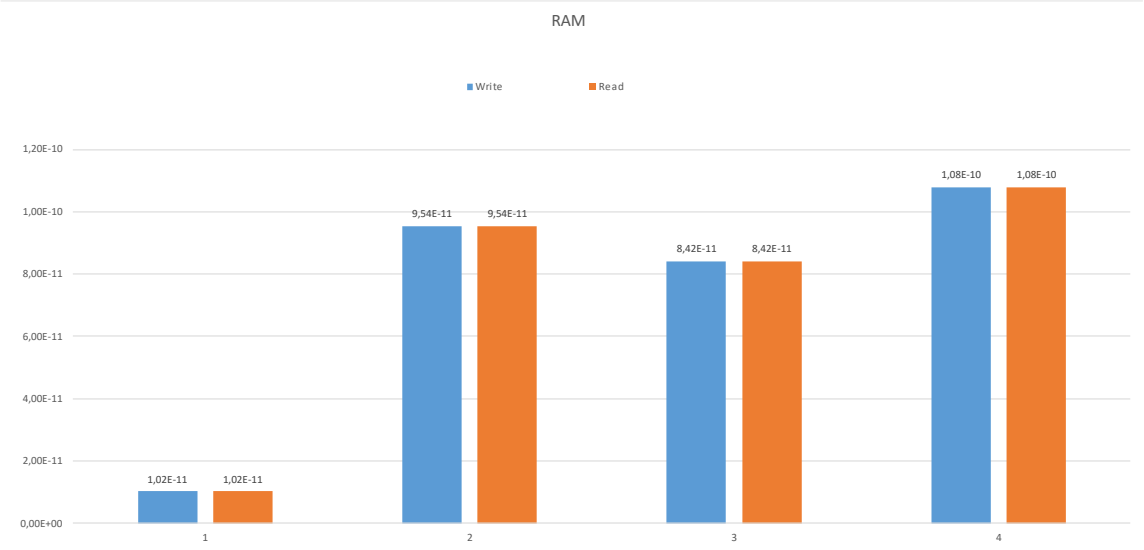


Рисунок 9. Зависимость погрешности измерений от числа испытаний LaunchNum.

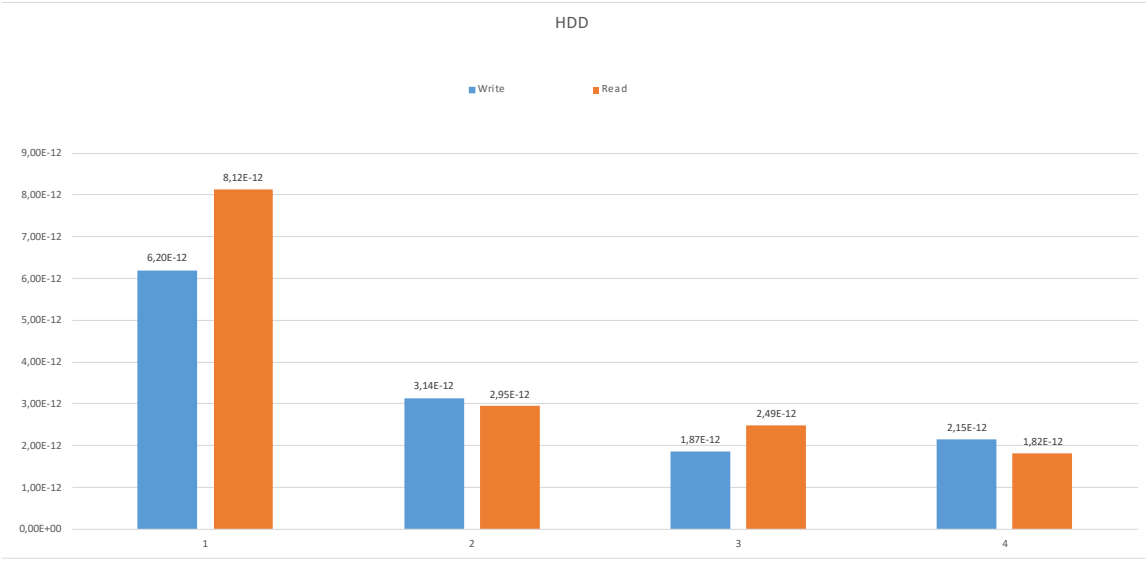


Рисунок 10. Зависимость погрешности измерений от числа испытаний LaunchNum.

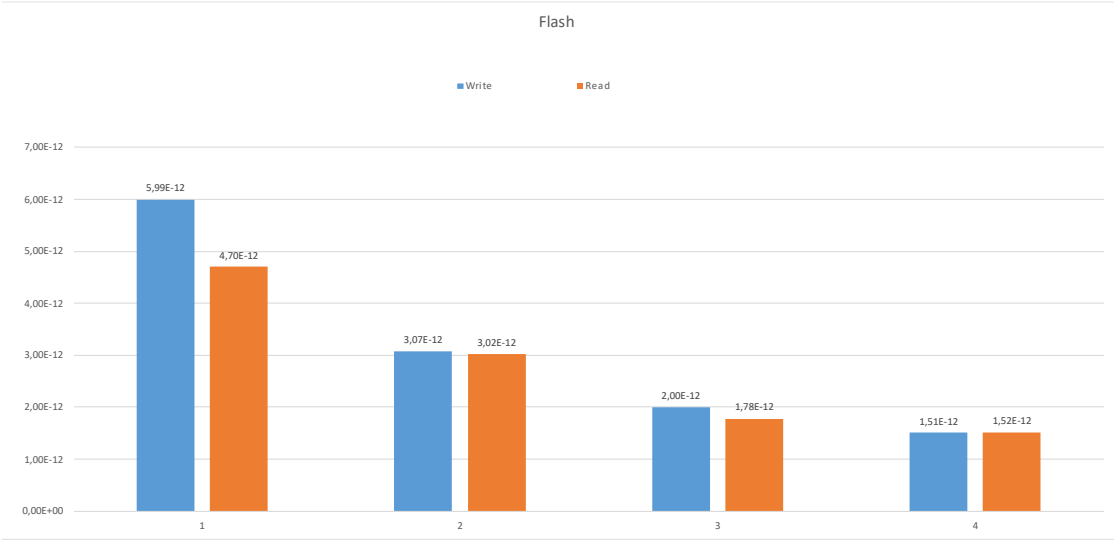


Рисунок 11. Зависимость погрешности измерений от числа испытаний LaunchNum.

Листинг

Файл mem_1.cpp

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <time.h>
#include "mem_t.hpp"

using namespace std;

int code_mem(string mem_t) {
    if (mem_t == "RAM") {
        return 0;
    }

    if (mem_t == "HDD") {
        return 1;
    }

    if (mem_t == "flash") {
        return 2;
    } else {
        return -1;
    }
}

int calc_size(int size_bl, string unit) {
    if (unit[0] == 'B') {
        return size_bl;
    }

    if (unit == "Kb" || unit == "KiB") {
        return 1024 * size_bl;
    }

    if (unit == "Mb" || unit == "MiB") {
        return 1024 * 1024 * size_bl;
    } else {
        return 0;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 4) {
        exit(EXIT_FAILURE);
    }

    string type_mem = (string) argv[1];
    string size_bl_str = (string) argv[2];
    int launch = atoi(argv[3]);
```

```
    string message = (string) argv[4];
    string ser = (string) argv[5];
    string unit = "";

    bool fl_ser = (ser == "1") ? 1 : 0;
    int size_bl_int = 0;
    string digit = "";
    int pos_unit = 0;

    for (int i = 0; i < size_bl_str.size() && size_bl_str[i] <= '9' &&
size_bl_str[i] >= '0'; i++, pos_unit++) {
        digit += size_bl_str[i];
    }

    for (int i = pos_unit; i < size_bl_str.size(); i++) {
        unit += size_bl_str[i];
    }

    size_bl_int = atoi(digit.c_str());
    unit = ((unit != "Kb" && unit != "KiB") && (unit != "Mb" && unit
!= "MiB")) ? "B" : unit;
    int N = calc_size(size_bl_int, unit);

    switch (code_mem(type_mem)) {
        case 0: {
            start_test(N, 0, launch, message, fl_ser);
            break;
        }
        case 1: {
            start_test(N, 1, launch, message, fl_ser);
            break;
        }
        case 2: {
            start_test(N, 2, launch, message, fl_ser);
            break;
        }
        default:
            cout << "Error type memory" << endl;
            exit(EXIT_FAILURE);
    }
}
```

Файл mem_t.cpp

```
#include "mem_t.hpp"

long double time_write = 0, time_read = 0;
vector<long double> vec_write, vec_read;
long double abs_w = 0, abs_r = 0;
```

```
void ram_test(int size_bl) {
    time_write = 0;
    time_read = 0;
    long double min_w = 10000, max_w = -10000;
    char *mas = new char[size_bl];
    struct timespec start, finish;

    for (int i = 0; i < size_bl; i++) {
        char r = rand() % 100 + 1;
        clock_gettime(CLOCK_REALTIME, &start);
        mas[i] = r;
        clock_gettime(CLOCK_REALTIME, &finish);
        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);

        if (t < min_w)  //мин время
            min_w = t;

        if (t > max_w)  //макс время
            max_w = t;

        time_write += t;
    }

    abs_w = max_w - min_w;
    abs_w /= size_bl;
    long double min_r = 10000, max_r = -10000;

    for (int i = 0; i < size_bl; i++) {
        clock_gettime(CLOCK_REALTIME, &start);
        mas[i];
        clock_gettime(CLOCK_REALTIME, &finish);
        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);
        time_read += t;

        if (t < min_r)
            min_r = t;

        if (t > max_r)
            max_r = t;

        time_read += t;
    }

    abs_r = max_w - min_w;
    abs_r /= size_bl;
    vec_write.push_back(time_write);
    vec_read.push_back(time_read);
```

```
    delete mas;
}

void hdd_ssd_test(int size_bl) {
    time_write = 0, time_read = 0;
    struct timespec start, finish;
    ofstream out("memory.dat", ios::binary);
//    ofstream out("/swapfile", ios::binary);
    long double min_w = 10000, max_w = -10000;

    for (int i = 0; i < size_bl; i++) {
        char buffer = rand() % 100 + 1;
        clock_gettime(CLOCK_REALTIME, &start);
        out.write((char *) &buffer, sizeof(buffer));
        clock_gettime(CLOCK_REALTIME, &finish);
        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);
        time_write += t;

        if (t < min_w)
            min_w = t;

        if (t > max_w)
            max_w = t;

        time_write += t;
    }

    long double med_w = (time_write) / size_bl;
    abs_w = max_w - min_w;
    abs_w /= size_bl;
    out.close();
    long double min_r = 10000, max_r = -10000;
    ifstream in("memory.dat", ios::binary);
//    ifstream in("/swapfile", ios::binary);

    for (int i = 0; i < size_bl; i++) {
        char buffer = rand() % 100 + 1;
        clock_gettime(CLOCK_REALTIME, &start);
        in.read((char *) &buffer, sizeof(buffer));
        clock_gettime(CLOCK_REALTIME, &finish);
        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);
        time_read += t;

        if (t < min_r)
            min_r = t;

        if (t > max_r)
            max_r = t;
```

```
        time_read += t;
    }

    abs_r = max_r - min_r;
    abs_r /= size_bl;
    in.close();
    vec_write.push_back(time_write);
    vec_read.push_back(time_read);
}

void flash_test(int size_bl, string path) {
    time_write = 0, time_read = 0;
    struct timespec start, finish;
    ofstream out(path + "memory.dat", ios::binary);
    long double min_w = 10000, max_w = -10000;

    for (int i = 0; i < size_bl; i++) {
        char buffer = rand() % 100 + 1;
        clock_gettime(CLOCK_REALTIME, &start);
        out.write((char *) &buffer, sizeof(buffer));
        clock_gettime(CLOCK_REALTIME, &finish);

        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);
        time_write += t;

        if (t < min_w)
            min_w = t;

        if (t > max_w)
            max_w = t;

        time_write += t;
    }

    abs_w = max_w - min_w;
    abs_w /= size_bl;
    out.close();
    long double min_r = 10000, max_r = -10000;
    ifstream in(path + "memory.dat", ios::binary);

    for (int i = 0; i < size_bl; i++) {
        char buffer = rand() % 100 + 1;
        clock_gettime(CLOCK_REALTIME, &start);
        in.read((char *) &buffer, sizeof(buffer));
        clock_gettime(CLOCK_REALTIME, &finish);

        long double t = 1000000000 * (finish.tv_sec - start.tv_sec) +
(finish.tv_nsec - start.tv_nsec);
```

```
        time_read += t;

        if (t < min_r)
            min_r = t;

        if (t > max_r)
            max_r = t;

        time_read += t;
    }

    in.close();
    abs_r = max_r - min_r;
    abs_r /= size_bl;
    vec_write.push_back(time_write);
    vec_read.push_back(time_read);
}

void start_test(int N, int type_mem, int launch, string message, bool
fl_ser) {
    ofstream out;
    out.open("info.csv", ios::app); //открыть для записи в конец файла
    out << message << endl;

    out << "MemoryType,"
        << "BlockSize,"
        << "ElementType,"
        << "BufferSize,"
        << "LaunchNum,"
        << "Timer,";

    out << "WriteTime,"
        << "AverageWriteTime,"
        << "WriteBandwidth,"
        << "AbsError(write),"
        << "RelError(write),";

    out << "ReadTime,"
        << "AverageReadTime,"
        << "ReadBandwidth,"
        << "AbsError(read),"
        << "RelError(read)" <<
        endl;

    int size_N = N; //змер в байтах

    switch (type_mem) {
        case 0: {
            for (int i = 0; i < launch; i++) { //по количеству тестов
                ram_test(N);
```

```
                write_result(type_mem, i + 1, N, out);
                if (fl_ser)
                    N += size_N;
            }
            break;
        }

        case 1: {
            for (int i = 0; i < launch; i++) {
                hdd_ssd_test(N);
                write_result(type_mem, i + 1, N, out);
                if (fl_ser)
                    N += size_N;
            }
            break;
        }
        case 2: {
            string path = "//media//tanya//SILICONPOWER//";


            for (int i = 0; i < launch; i++) {
                flash_test(N, path);
                write_result(type_mem, i + 1, N, out);
                if (fl_ser)
                    N += size_N;
            }
            break;
        }
    }

    out.close();
}

void write_result(int type_mem, int launch, int bl_size, ofstream
&out) {
    double AverageWrT = 0;
    for (int i = 0; i < vec_write.size(); i++) {
        AverageWrT += vec_write[i] / 1000000000;  //среднее значение
    }

    AverageWrT /= launch;
    double AverageReT = 0;

    for (int i = 0; i < vec_write.size(); i++) {
        AverageReT += vec_read[i] / 1000000000;
    }

    AverageReT /= launch;
    double WriteBand = ((bl_size / AverageWrT) / 1024) / 1024;
//пропускная способность памяти
```

```
    double ReadBand = ((bl_size / AverageReT) / 1024) / 1024;
//пропускная способность памяти
    double med_w = ((time_write / 1000000000) / bl_size);
    abs_w /= 1000000000;
    long double rel_w = (abs_w / med_w) * 100;
    double med_r = ((time_read / 1000000000) / bl_size);
    abs_r /= 1000000000;
    long double rel_r = (abs_r / med_r) * 100;
    string type_mem_str = str_mem(type_mem);

    out << type_mem_str << "," << bl_size << " байт,"
        << "char,"
        << "1 байт," << launch << ","
        << "clock_gettime," << time_write / 1000000000 << " сек," <<
AverageWrT << " сек,"
        << WriteBand * pow(10, 3) << " МБ/с," << abs_w << " сек," <<
rel_w << "%,"
        << time_read / 1000000000 << " сек," << AverageReT << " сек,"
<< ReadBand * pow(10, 3) << " МБ/сек,"
        << abs_r << " сек," << rel_r << "%" << endl;

}

string str_mem(int mem_t) {
    if (mem_t == 0) {
        return "RAM";
    }

    if (mem_t == 1) {
        return "HDD";
    }

    if (mem_t == 2) {
        return "flash";
    } else {
        return "";
    }
}
```

Файл mem_t.hpp

```
#pragma once
#include <time.h>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <vector>
#include <cmath>

using namespace std;
```

```
void ram_test(int size_bl);
void write_result(void);
void hdd_ssd_test(int size_bl);
void flash_test(int size_bl, string path);
void write_result(void);
void start_test(int N, int type_mem, int launch, string message, bool
fl_mes);
void write_result(int type_mem, int launch, int bl_size, ofstream
&out);
string str_mem(int mem_t);
```

Файл main.cpp

```
#include <bits/stdc++.h>

using namespace std;

enum {
    BS = 8
};

void DGEMM(double **a, double **b, double **c, long long N) {
    for (long long i = 0; i < N; i++) {
        for (long long j = 0; j < N; j++) {
            for (long long k = 0; k < N; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void DGEMM_OPT1(double **a, double **b, double **c, long long N) {
    for (long long i = 0; i < N; i++) {
        for (long long k = 0; k < N; k++) {
            for (long long j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void DGEMM_OPT2(double **a, double **b, double **c, long long N) {
    int i0 = 0, k0 = 0, j0 = 0, str1 = 0, str2 = 0, n = N;
    double *a0 = NULL, *b0 = NULL, *c0 = NULL;

    for (int i = 0; i < n; i += BS) {
        for (int j = 0; j < n; j += BS) {
            for (int k = 0; k < n; k += BS) {
```

```
                for (i0 = 0, c0 = &c[i][j], a0 = &a[i][k], str1 = i+1;
i0 < BS;++i0, c0 = &c[str1][j], a0 = &a[str1][k], str1+=1){
                    for (k0 = 0, b0 = &b[k][j], str2 = k+1;k0 < BS;
++k0, b0 = b[str2], str2 += 1) {
                        for (j0 = 0; j0 < BS; ++j0){
                            c0[j0] += a0[k0] * b0[j0];
                        }
                    }
                }
            }
        }
    }
}


void rand_mass(double **a, double **b, long long n) {
    srand (time(NULL));
    for(long long i = 0; i < n; ++i){
        for(long long j = 0; j < n; ++j){
            a[i][j] = rand() % 100;
            b[i][j] = rand() % 100;
        }
    }
}

void series_func(string multiply, int left, int right) {
    clock_t start, finish;
    if (multiply == "opt0") {
        start = clock();
        ofstream out("result_opt0.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM(a, b, c, k);
            finish = clock();
```

```
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC
<< endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;

            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
    } else if (multiply == "opt1") {
        start = clock();
        ofstream out("result_opt1.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM_OPT1(a, b, c, k);
            finish = clock();
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC
<< endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;
            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
```

```
    } else if (multiply == "opt2") {
        start = clock();
        ofstream out("result_opt2.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM_OPT1(a, b, c, k);
            finish = clock();
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC
<< endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;
            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
    }
}

void single_func(string multiply, int n) {
    if (multiply == "opt0") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
```

```
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        rand_mass(a, b, n);
        DGEMM(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    } else if (multiply == "opt1") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }
        rand_mass(a, b, n);
        DGEMM_OPT1(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    } else if (multiply == "opt2") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
```

```
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }
        rand_mass(a, b, n);
        DGEMM_OPT2(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    }
}
int main(int argc, char** argv) {
    if(argc < 3){
        cout << "Слишком мало аргументов";
        return 1;
    } else if((string)argv[1] == "single"){
        single_func((string)argv[2], atoi(argv[3]));
    } else if((string)argv[1] == "series") {
        if (argc < 5) {
            cout << "Слишком мало аргуметов для серии";
            return 1;
        } else{
            series_func((string) argv[2], atoi(argv[3]),
atoi(argv[4]));
                }
        }
}
```